SCML Syntax

SCML Syntax

Define Objects

As in any XML based markup language objects are defined using tags.

As an example, a group with no properties is defined like this:

```
<Group></Group>
```

Or alternatively like this:

```
<Group />
```

For all object tags available refer to the SCML Tags section.

Define Children

Children of an object are defined between its opening and closing tags.

As an example, define a group as child of another group like this:

```
<Group>
   <Group />
</Group>
```

Define Properties

Properties are defined as attributes on the tag representing the object. The syntax to define object properties depends on the property type.

Properties of Built-in Simple Types

Built-in simple types are: float, int, bool, string, and all enumeration types.

As an example, a PointSprite with some simple type properties is defined like this:

```
<PointSprite
  Name ="PointSprite"

  PointSize ="100"
  IsRenderingEnabled ="False">
</PointSprite>
```

Properties of Built-in Complex Types with Simple Textual Representations

Simple textual representations are the ones commonly accepted for comples types and which are used in the Properties panel.

As an example, a PointSprite with simple and also complex type properties like Vector3, Matrix4 and Color is defined like this:

```
<PointSprite
  Name ="PointSprite"

  Position ="0;1;0" Rotation ="0;1;0" Scale ="10;1;1"
  Color
="0.5;0.5;0.5;0.5"
  PointSize ="100"

  Transform ="1;0;1;0 | 0;1;0;0 | 0;0;1;0 | 0;0;0;1">

</PointSprite>
```

Properties of Built-in Complex Types with no Simple Textual Representation

Properties of types that are too complex to represent as attributes are defined as child tags of the form <Object.Property>. These tags must be defined right after the opening tag, before any children are defined.

As an example, the complex Appearance property of a Group object is defined like this:

```
<Group>
  <Group.Appearance>
    <Appearance IsShaderProgramSet ="False">



  </Group.Appearance>

</Group>
```

This syntax can of course also be used for complex types with simple textual representation, in order to increase the readability. Refer to following example using this syntax for a Color:

```
<Material>
  <Material.Ambient>
    <Color Red ="0.5588974" Green ="0.5447855454" Blue ="0.5478851" Alpha ="0.255454145">



  </Material.Ambient>

</Material>
```

Collection Properties

Like complex properties, collections are in a tag of the form <Object.CollectionProperty> that must be placed right after the opening tag of the object, and before any children are defined.

The objects to be added to the collection are defined between the opening and closing collection property tag.

As an example, an appearance object with a collection of two textures is defined like this:

```
<Appearance>
  <Appearance.Textures>
    <Texture> </Texture>
    <Texture> </Texture>
  </Appearance.Textures>
</Appearance>
```

Dynamic Properties

Dynamic properties (like those of widgets and render targets) are defined like any other property but using the form DynamicProperty.

As an example, a widget with two dynamic properties is defined like this:

```
 <Scene.Widgets>
  <Widget
Name="LabelWidget"TemplateName="../../#Config/Widget:LabelWidget"WidgetNodeName="./CubScene/Gr
oup:RootNode/Mesh:AvantgardeSimpleCube">

    <Widget.DynamicProperties>

      <DynamicProperty Name="Enabled" Type="Bool" Value="True">
      </DynamicProperty>

      <DynamicProperty Name="TextColor" Type="Color" Value="0,0,0,1">

      </DynamicProperty>

    </Widget.DynamicProperties>
  </Widget>

</Scene.Widgets>
```

Revision #2
Created 2023-02-15 08:03:45 UTC by Tsuyoshi.Kato
Updated 2023-11-27 09:05:01 UTC by Tsuyoshi.Kato